Draft Topic Area Levels

REUSE BREAKOUT SESSION – Oct. 23, 2008

1 Portability

The levels below are defined relative to the users most likely to benefit from the software and the contexts that they are most likely to have.

RRL 1 – The software is not portable
No source code or instructions for customization are provided. Executable binaries are provided and there are known severe limitations for running it on the hardware or operating system. There is only minimal information on installation or use. There is no information on porting to another platform or application.

RRL 2 – Some parts of the software may be portable
Some source code is provided with some internal and external documentation. Binaries are provided and there is some documentation on how to install the software. There is no useful information on porting. Porting is prohibitively expensive, but some portions (e.g. modules, functions) of the code may be portable.

RRL 3 – The software is only portable with significant costs
The complete source code is available, without external dependencies that are portable, but the software cannot be ported without significant changes to the software or the target context. Documentation on porting the code to another platform or application is missing or deficient. Porting would not be practical or cost effective.

RRL 4 – The software may be portable at a reasonable cost
The cost benefits of using the software slightly outweigh the cost of developing new software. Documentation is barely sufficient, but may contain some useful information on porting to another platform or application. Porting will nonetheless require significant effort. Only at this level is it generally worth considering porting the software.

RRL 5 – The software is moderately portable
The software can be ported with only relatively small changes necessary to the context or the software itself. Documentation on porting exists and is complete, but requires considerable effort and expertise. Some rudimentary understanding of the underlying software or the target system may be necessary.

RRL 6 - The software is portable
The software can be ported to most major systems without modification. The documentation, however, addresses porting to a large number of systems that are identified. Any modifications needed to port the software to these systems are well described in the documentation and would be relatively easy to implement.

RRL 7 – The software is highly portable
The software can be ported to all but the most obscure or obsolete systems without modification. The documentation is complete and thorough. No changes to the software are necessary and the effort to port the software is minimal.

RRL 9 – The software is completely portable
The software can be ported to all systems since it runs on an application layer rather than on the underlying operating system layer. Such software is written in languages Java, C#, etc. In theory at least, the software will run on any system in which the appropriate application layer has been installed.

2 Extensibility

RRL 1 – No ability to extend or modify program behavior

Source code is not available; execution parameters cannot be changed, and/or it is not possible to extend the functionality of the software, even for application contexts similar to the original application domain.

RRL 2 – Very difficult to extend the software system, even for application contexts similar to the original application domain
The software was not designed with extensibility in mind. While some level of documentation and/or source code is available, it is extremely difficult to extend the software. For cases where source code is available, the logical flow of code may be hard to follow, with few (if any) comments, and little to no cohesion.

RRL 3 – Extending the software is difficult, even for application contexts similar to the original application domain
Minimal consideration to extensibility is included in the design, through use of methods such as object-oriented design or other tools which provide logical cohesion. Where source code is available, the software has some structure, but may have a high number of independent logical paths, minimal comments and documentation, and/or a low degree of cohesion.

RRL 4 – Some extensibility is possible through configuration changes and/or moderate software modification
Consideration to extensibility to some range of application contexts is included in the design though means such as a) use of configuration files, b) isolation of configuration parameters and constants in clearly identified sections of source code (distinct from logic and display code), c) some documentation of the effects of changes to these parameters and the allowed values for these parameters, and/or d) effective use of programming practices designed to enable reuse, such as object oriented design.

RRL 5 – Consideration for future extensibility designed into the system for a moderate range of application contexts; extensibility approach defined and at least partially documented

The procedures for extending the software are defined, whether by source code modification (e.g., object-oriented design) or through the provision of some type of extension functionality (e.g., callback hooks or scripting capabilities). Where source code modification is part of the extension plan, the software is well-structured, has a moderate to high level of cohesion, and has configuration elements clearly separated from logic and display elements. Internal and external documentation are sufficient to allow an experienced programmer to understand program flow and logic with moderate effort.

RRL 6 – Designed to allow extensibility across a moderate to broad range of application contexts, provides many points of extensibility, and a thorough and detailed extensibility plan exists
The extensibility capability for the software is well defined, sufficient to enable an experienced developer generally familiar with the project to extend the software. That documentation should include clear information about the range of application contexts to which the software can be extended as well as potential limitations on expansion.

RRL 7 – Demonstrated to be extensible by an external development team in a similar context
The software has been extended and applied to a similar application context to the original. This extension may have been done by an external team using extension documentation, by may have involved substantial assistance from the original development team members.

RRL 8 – Demonstrated extensibility on an external program, clear approach for modifying and extending features across a broad range of application domains
The software has been extended by at least one group of users outside the original development group using existing documentation and with no assistance from the original development team.

RRL 9 – Demonstrated extensibility in multiple scenarios, provides specific documentation and features to build extensions which are used across a range of domains by multiple user groups
The software is regularly extended externally by users across multiple applications using available documentation. There may be a library available of user-generated content for extensions.

3 Documentation

[Consider black box in some of places where it is not explicitly mentioned.]

RRL 1 – Little or no internal or external documentation available
Source code is available, with little or no useful internal or external documentation.

RRL 2 – Partially to fully commented source code available
Source code is available and fully commented, but no other documentation is provided. It may be challenging for a good programmer to determine how to reuse the software.

RRL 3 – Basic external documentation for sophisticated users available
For example, a README file, a “man” page, or command line usage examples. This type of documentation would be sufficient for a sophisticated user to figure out how to use the software, but probably not a general user.

RRL 4 – Reference manual available
Reference manual provides complete documentation on use of the software, but may not be easily approached or accessed by general users. Some documentation relevant to customization is available.

RRL 5 – User manual available
User manual allows a “normal” or general user to understand how to use and possibly customize aspects of the software.

RRL 6 – Tutorials available
Step-by-step walkthroughs of how the software is customized and used in various scenarios, demos, etc. This makes it very easy to understand/teach the software and use it in a new project.

RRL 7 – Interface guide available
Documentation describes how to customize and interface the software with other software, programmatic interfaces, APIs, etc., so that it can more easily be embedded in a larger system.

RRL 8 – Extension guide and/or design/developers guide available
An extension guide provides information on how to customize and add to the software, add plugins and the like, use internal programming “languages”, etc. A design/developers guide provides a description of internals, design documentation, internal documentation, etc. that is sufficient for someone “skilled in the art” to contribute to the development of the software or take over maintenance of the software.

RRL 9 – Documentation on design, customization, testing, use, and reuse is available
All stages of the software engineering lifecycle are fully documented. This includes design and review artifacts, testing artifacts, customization, and regression tests. The documentation provided is easy to read/access and is appropriate for different categories of users.

4 Support

RRL 1 – No support available
The original developer of the code is not known, not locatable, or is refusing support.

RRL 2 – Minimal support available
There is known contact information available for the original developer(s) and they are willing to provide minimal, occasional support.

RRL 3 – Some support available
Contact information is available and there is a willingness to provide some support infrequently, without guarantees. This may include things such as providing makefiles or different flavors of the code for different contexts.

RRL 4 – Moderate systematic support is available
Latest updates/patches are usually made available. Support is available, but may be intermittent.

RRL 5 – Support provided by an informal user community
There is an informal user community that provides answers, for example, via a Web site FAQ.

RRL 6 – Formal support available
Support is centralized in a web site containing relevant resources, answers to FAQ, and other useful information.

RRL 7 – Organized/defined support by developer available
There is organized and defined support by the developer with email/telephone help desk and links to case studies and other relevant information. No continuity of support implied.

RRL 8 – Support available by the organization that developed the asset
The support is by an organization and is well defined with frequent updates, releases, etc., and help desk. Continuity of support is implied. Support may be free or fee-based and may be offered by a third party.

RRL 9 – Large user community with well-defined support available
This may include resources such as a help desk, a Web site for the latest information, an active discussion group willing to answer questions, frequent patches and updates as well as consolidation of changes by the community. One example would be the Linux operating system.

5 Packaging

RRL 1 – Software or executable available only, no packaging
Inadequate or no documentation and no auto-build/install facility is available.

RRL 3 – Detailed installation instructions available
System includes auto-build feature, but is built for a particular operating system.

RRL 5 – Software is easily configurable for different contexts
For example, locations of resources (files, directories, URLs) are configurable. All configuration-specific information is centralized.

RRL 7 – OS-detect and auto-build for supported platforms available
Operating system detection configuration files are available. Packaging includes auto-build for supported OS platforms and suite of regression tests for platform-specific testing.

RRL 9 – Installation user interface provided
A user interface guides the installer through all steps needed to build, configure, and install the software.

6 Intellectual Property Issues

Developer in this section is either the person who developed the software or the organization responsible for developing the software. This needs to be reduced to fewer levels with more attention to the process (steps) of releasing the software from the developer's organization. 1) No rights, 2) in process, 3) determined.]

RRL 0 – No developer has been identified, rights are undetermined
Product developers have not been identified and so the rights are undetermined.

RRL 1 – Product developers have been identified, but no rights have been determined.
Product developers have been identified and their responsibilities have been determined, but they have not considered or determined the rights for the product.

RRL 2 – Developers are discussing rights that comply with their organizational policies.
Relevant policies of developers have been reviewed for applicability to intellectual property rights, but no agreements have been proposed. Rights are not specified.

RRL 3 – Rights agreements have been proposed to developers.
Each developer has received a draft intellectual property rights agreement that would result from cooperative activities with other developers. Rights are not specified.

RRL 4 – Developers have negotiated on rights agreements.
Developers have reviewed proposals from each of the other developers and have proposed an agreement that addresses any potential conflicts in the proposed intellectual property rights and responsibilities for development. A limited rights statement has been drafted and developers may be contacted to negotiate rights for reuse.

RRL 5 – Agreement on ownership, limited reuse rights, and recommended citation.
Developers have agreed on proposed ownership, limited intellectual property rights for reuse, and responsibilities. Order of developers' names, recommended citation, and agreements have been formalized. Developers may be contacted to obtain formal statements on restricted rights for reuse.

RRL 6 – Developer list, recommended citation, and rights statements have been drafted .
Agreements on development responsibilities, the list of developers, a recommended citation, and intellectual property rights statements, offering limited rights for reuse have been drafted and are included in package. Developers may be contacted to obtain formal statements on restricted rights or to negotiate additional rights.

RRL 7 – Developer list and limited rights statement included in product prototype.
A list of developers, recommended citation, and intellectual property rights statements, including copyright or ownership statements, are embedded in the source code of the product, in the documentation, and in the expression of the software upon execution. These include any legal language that has been approved by all parties or their representatives, machine-readable code expressing intellectual property, and concise statements in language that can be understood by laypersons, such as a pre-written, recognizable license. Brief statements are available describing limited rights, restrictions, and conditions for reuse. Developers may be contacted to negotiate additional rights.

RRL 8 – Recommended citation and intellectual property rights statement included in product.
All parties have reviewed the list of developers, recommended citation, and intellectual property rights statements, including limited rights for reuse, in the product to ensure that all interests are represented and that the statements conform to their institutional policies and agreements. Brief statements are available describing unrestricted rights and any conditions for reuse. Developers may be contacted to obtain formal rights statements.

RRL 9 – Statements describing unrestricted rights, recommended citation, and developers embedded into product.
Multiple statements are embedded into the product describing unrestricted rights and any conditions for reuse, including commercial reuse, and the recommended citation. The list of developers is embedded in the source code of the product, in the documentation, and in the expression of the software upon execution. The intellectual property rights statements are expressed in legal language, machine-readable code, and in concise statements in language that can be understood by laypersons, such as a pre-written, recognizable license.

General comments:

- Consider if the following areas are addressed in the IP statements:
    - Who is allowed to use the software (as a user)?
    - Who can "reuse" the software as a component of their code?
    - Does reusing the software obligate the reuser to certain actions (make their modification source available or open source, make their entire code open source)?
    - Can parts of the software be reused freely, while others are restricted?

7 Standards Compliance

[This section should be reviewed by the standards group. This section refers to both software and/or the software development process.]

RRL 1 – No standards compliance
Neither the software nor the software development process adhere to any identified standards other than those inherent in the software languages employed.

RRL 2 – No standards compliance beyond best practices
The software and software development process adhere, at least in part, to some common best practices, but do not identify or claim compliance with any recognized standard.

RRL 3 – Some compliance with local standards and best practices
The software and software development process comply with standards and best practices defined locally by the development organization.

RRL 4 – Standards compliance, but incomplete and untested
The software and software development process attempt to comply with recognized standards, but without verification. Standards compliance is thus untested and may not be complete.

RRL 5 – Standards compliance with some testing
The software and software development process comply with recognized standards, but verification of compliance is incomplete. Standards compliance may not be followed by all components.

RRL 6 – Verified standards compliance with proprietary standards
The software and software development process comply with specific and proprietary standards (such as Windows GUI) and compliance with those standards has been verified through testing.

RRL 7 – Verified standards compliance with open standards
The software and software development process comply with specific open standards and compliance with those standards has been verified through testing.

RRL 8 – Verified standards compliance with recognized standards
The software and software development process comply with internationally recognized standards such as W3C, XML, XHTML, WAI, IP for Web; or ANSI/ISO (C/C++), JCP (Java), for software; and CMMI, IEEE Software Engineering Standards for development process. Standards compliance has been verified through testing, but not by an independent testing organization.

RRL 9 – Independently verified standards compliance with recognized standards
The software and software development process comply with internationally recognized standards. Independent and documented standards compliance verification is included with the software. The development organization maintains standards compliance in its development process through regular testing and certification from an independent group.

Notes:
1. Proprietary standards such as Windows GUI are very non-conducive to universal reuse
2. openstandards.org has a great listing of standards relevant to the open source community

8 Verification and Testing

[Define the terms verification and testing.]

RRL 1 – No testing performed.
Ideas have been translated into software development. Examples might include studies of development languages, prototype, diagram of interface. Requirements have not been verified, and there is no formal test mechanism in place.

RRL 2 – Software application formulated and unit testing performed.
Software application compiles, and executes with known inputs. For example, a prototype application where there is no testing or validation to support the software, but only testing to demonstrate a prototype. Requirements may not be finalized yet, or overall testability of the software determined.

RRL 3 –Testing includes testing for error conditions and handling of unknown input.
Software applications have been 'white box' tested. This includes both known and unexpected inputs to the application. This level of testing has been incorporated into the build and/or deployment mechanism.

RRL 4 – Software application demonstrated in a laboratory context.
Following successful testing of inputs and outputs, the testing has integrated an application to establish that the "pieces" will work together to achieve concept-enabling levels. This validation has been devised to support the concept that was formulated earlier, and is consistent with the requirements of potential system applications. The validation is relatively "low-fidelity" compared to the eventual system – it could be composed of ad hoc discrete components in a laboratory; for example, an application tested with simulated inputs.

RRL 5 – Software application tested and validated in a laboratory context.
The fidelity of the software application testing has not been demonstrated. The software application must be integrated with reasonably realistic supporting elements so that the total application (component level, sub-system level, or system level) can be tested in a "simulated" or somewhat relevant context. At this level, issues such as scalability, load testing, and security are addressed when applicable.

RRL 6 – Software application demonstrated in a relevant context.
The fidelity of the software application testing has not been demonstrated. The software application must be integrated with existing elements and interfaces so that the total application (component level, sub-system level, or system level) can be tested and validated in a relevant context. At this level, issues such as number of users and operational scenarios, as well as load testing and security are addressed if applicable.

RRL 7 – Software application tested and validated in a relevant context.
The software application testing meets the requirements of the application that apply to the software when it is to be delivered or installed. The software application has been tested in the lab so that the application can be validated as if the software were delivered for use in another context. At this level, all issues have been resolved regarding security and operational scenarios.

RRL 8 – Software application “qualified” through test and demonstration (meets requirements) and successfully delivered.
The software has passed testing and meets all requirements of the software, with the additional testing of the software delivery and installation for various applications.

RRL 9 – Actual software application tested and validated through successful use of application output.
Demonstrable that for any application of the software, testing shows the software meets all defined requirements. [Talk about the rigorous testing process itself, against a set of testable requirements. This could be independent.]

General comments: Business Readiness Rating (http://www.openbrr.org/wiki/index.php/Downloads) information on Performance includes some testing information that may help with this topic.

9 Modularity

RRL 1 – Not designed with modularity
Research or prototype-grade code written with no designs for organizing code in terms of functionality for modularity or reuse.

RRL 3 – Modularity at major system or subsystem level only
No clear distinctions between generic and solution-specific functionality; few internal functions accessible by external programs (i.e., closed architecture), limited distinction between visible functions; code is organized into a primary system that provides general functionality and one or two subsystems that each provide multiple, unrelated, functions, requiring additional modularity?; code within each module contains many independent logical paths.

RRL 5 – Partial segregation of generic and specific functionality
Top to bottom structuring into individual components that provide functions or services to outside entities (i.e., open architecture); internal functions or services documented, but not consistently; modules have been created for generic functions, but modules have not been created for all of the specified functions; code within each module contains many independent logical paths.

RRL 7 – Clear delineations of specific and reusable components;
Organization of all components into libraries or service registries; consistent documentation of all libraries as APIs or standard web service interfaces; modules have been created for all specified functions and organized into libraries with consistent features within interfaces; code within each module contains many independent logical paths.

RRL 9 – All functions and data encapsulated into objects or accessible through web service interfaces
All functions and data encapsulated into objects or accessible through web service interfaces; consistent error handling; use of generic extensions to program languages for stronger type checking and compilation-time error checking; services available externally, e.g., in "third-party" service workflows; code within each module contains few independent logical paths.

Notes:
Business Readiness Rating (http://www.openbrr.org/wiki/index.php/Downloads) information on Scalability may relate to extensibility and/or modularity.

General Topic Level Comments

Business Readiness Rating (http://www.openbrr.org/wiki/index.php/Downloads) information on Security may help us factor that into our topic levels, since we got feedback suggesting we consider the issue of security. Their quantitative levels may help in some of our topic areas, too, noted above.

The verb tense should be consistent across all topic areas and across all levels.

Some of the more general feedback received may need some explanatory text to address them. Trying to fit it directly into the topic area levels may not work very well. Examples:

- Use vs. reuse – how we classify the difference and how much of each we're considering
    - Can use the WG definitions posted on the portal web site at http://www.esdswg.org/softwarereuse/Resources/library/reuse-definitions/
    - "In general, if you have acquired (or used) a software development asset from someplace else that otherwise you would have written yourself then you have experienced the benefit of reuse."
- Is open source software a pre-requisite for reuse?
    - My feeling is no, you can do black-box reuse of a compiled library or executable, for example.
    - However, reuse can be easier with open source software, and perhaps open source software can automatically start at, say, RRL 2 to reflect that.
- Defining the target audience (e.g., developers or managers?)
    - I think we're trying to cover both.
    - Developers might prefer to use the detailed topic levels.
    - Managers might prefer to use the overall summary levels.
- Different types of reuse – black box, white box, VM-ware, reuse after long periods of time, etc.
    - I think we need to account for at least black and white box reuse.
    - Perhaps reuse in virtual machines is a factor within portability? I'm not sure it's something that cuts across all topic areas.

    - Reuse after long periods of time seems to be more of a preservation or life cycle issue.
- Cost and risk are considered important; how do these factor in?
    - To some extent, these are implicitly covered by the levels: lower levels carry more cost and risk than higher levels.
    - Should reuse readiness, which seems more of a technical concept, explicitly include factors like cost and risk, which may depend on the project reusing the software?

Draft Reuse Readiness Levels

RRL 1 – No reusability; the software is not reusable.
Little is provided beyond limited source code or pre-compiled, executable binaries. There is no support, contact information for developers or rights for reuse specified, the software is not extensible, and there is inadequate or no documentation.

RRL 2 – Initial reusability; software reuse is not practical.
Some source code, documentation, and contact information are provided, but these are still very limited. Initial testing has been done, but reuse rights are still unclear. Reuse would be challenging and cost-prohibitive.

RRL 3 – Basic reusability; the software might be reusable by skilled users at substantial effort, cost, and risk.
Software has some modularity and standards compliance, some support is provided by developers, and detailed installation instructions are available, but rights are unspecified. An expert may be able to reuse the software, but general users would not.

RRL 4 – Reuse is possible; the software might be reused by most users with some effort, cost, and risk.
Software and documentation are complete and understandable. Software has been demonstrated in a lab on one or more specific platforms, infrequent patches are available, and intellectual property issues would need to be negotiated. Reuse is possible, but may be difficult.

RRL 5 – Reuse is practical; the software could be reused by most users with reasonable cost and risk.
Software is moderately portable, modular, extendable, and configurable, has low-fidelity standards compliance, a user manual, and has been tested in a lab. A user community exists, but may be a small community of experts. Developers may be contacted to request limited rights for reuse.

RRL 6 – Software is reusable; the software can be reused by most users although there may be some cost and risk.
Software has been designed for extensibility, modularity, and portability, but software and documentation may still have limited applicability. Tutorials are available, and the software has been demonstrated in a relevant context. Developers may be contacted to obtain formal statements on restricted rights or to negotiate additional rights.

RRL 7 – Software is highly reusable; the software can be reused by most users with minimum cost and risk.
Software is highly portable and modular, has high-fidelity standards compliance, provides auto-build installation, and has been tested in a relevant context. Support is developer-organized, and an interface guide is available. Software and documentation are applicable for most systems. Brief statements are available describing limited rights for reuse and developers may be contacted to negotiate additional rights.

RRL 8 – Demonstrated reusability; the software has been reused by multiple users.
Software has been shown to be extensible, and has been qualified through test and demonstration. An extension guide and organization-provided support are available. Brief statements are available describing unrestricted rights for reuse and developers may be contacted to obtain formal rights statements.

RRL 9 – Demonstrated reusability; the software is being reused by many classes of users over a wide range of systems.
Software is fully portable and modular, with all appropriate documentation and standards compliance, encapsulated packaging, a GUI installer, and a large support community that provides patches. Software has been tested and validated through successful use of application output. Multiple statements describing unrestricted rights for reuse and the recommended citation are embedded into the product.